Compact Signature Representation

EIP-2098

Richard Moore

`@ricmoo`

👀 PEEPanEIP – February 8, 2022

What are Signatures?

Signatures are part of a *proof system*, which are used extensively through-out Ethereum

- Every Ethereum Transaction includes a Signature, which proves an account has agreed to the transaction
- Many smart contracts can directly process a Signature, which allows it to validate proofs that an account has agreed to some action

Signatures: An Anatomy

example signature

```
{
  r: "0xa4d0ca120293144cd29d20bf6b3f8c5a45a45720ff94935f8d00a329de504264",
  s: "0x1cfa6be18e8c379cfc22764d65b92ff49c09506c122a79d07e40c5234db9e9ef",
  v: 28
}
```

R - The ***x*** co-ordinate of a *Challenge* or *Reference point,* effectively created at random

S - The proof; for a given ***R***, only the account owner can calculate this

V - Recovering a public key requires the full *Challenge* (***R*** is **only** the ***x*** co-ordinate), which for any ***x*** has two valid values for ***y***; the value 27 and 28 are used to specify which to use

Canonical Signatures

The Homestead hardfork (EIP-2) introduced *canonical signatures*, which put a restriction on the allowed values for ***S***

- This prevents a form of transaction malleability, which didn't really impact Ethereum in any significant way, but was certainly a nice-to-have
- Had a profound (positive) impact, à la this EIP; it frees up a bit in a Signature!!

Canonical Signatures: Consequences

```
// Hex
s: 0x1cfa6be18e8c379cfc22764d65b92ff49c09506c122a79d07e40c5234db9e9ef

// Binary
S: 0001 1100 1111 1010 0110 1011 1110 0001 1000 ...
   ^
   \------ This is always 0
```

Since the ***S*** must always be less than half the *order of the secp256k1 curve*, the highest bit must always be 0.

Putting it all together...

Since the value of ***V*** is always 27 or 28 and we have an extra bit that must always be 0 in the signature...

```
// Start with the same S
compactS = S

// Map v=27 => 0 and v=28 => 1, and place that value in the top bit
If (v == 28) {
  compactS |= (1 << 255)
}

Signature = { R, compactS }
```

Size Comparison: Solidity

```
// bytes - 160 bytes of calldata
function withBytes(bytes signature);

// separate parameters - 96 bytes of calldata
function withParams(bytes32 r, bytes32 s, uint8 v);

// EIP-2098 compact - 64 bytes of calldata
function withCompact(bytes32 r, bytes32 compactS);
```

In Solidity calldata, *compact signature representation* saves 60% over a bytes and 33% over separate parameters.

Size Comparison: Transactions

With *EIP-2718 Transaction Envelopes*, all new transaction types use traditional signatures; with EIP-2098 compact signatures, all future transaction types would be reduced by an entire byte.

```
// At 14 TPS:
> 24 * 60 * 60 * 14
1209600

// This would save 1.2 megabytes per day, that *all* nodes
// must retain.
```

Backwards Compatibility

Since traditional signatures are 65 bytes and EIP-2098 compact signatures are 64 bytes there is no collision in the two representations.

Questions?

EIP: `https://eips.ethereum.org/EIPS/eip-2098`

Discussion: `https://github.com/ethereum/EIPs/issues/2440`

Richard Moore

`@ricmoo`